"Express Mail" mailing label number EV 331533665 US

Date of Deposit: September 3, 2003

ATTORNEY DOCKET NO. 14118US02

PROCESSING SYSTEM WITH COMPONENT ARCHITECTURE PLATFORM SUPPORT

CROSS-REFERENCE TO RELATED APPLICATIONS/INCORPORATION BY REFERENCE

[0001] The present application claims the benefit of U.S. Provisional Application Serial No. 60/412,858 filed on 23 September 2002, which in turn claims the benefit of PCT Application having Publication No. WO/02/41147 A1, PCT Serial No. PCT/US01/44034, filed 19 November 2001, which in turn claims the benefit of U.S. Provisional Application Serial No. 60/249,606 filed 17 November 2000, the complete subject matter of each of which is hereby incorporated herein by reference in its entirety.

[0002] The present application also incorporates herein by reference in its entirety, co-pending U.S. Application Serial No. _____ entitled "EMBEDDED SYSTEM EMPLOYING COMPONENT ARCHITECTURE PLATFORM", filed 3 September 2003.

FEDERALLY SPONSORED RESEARCH OR DEVELOPMENT

[0003] [Not Applicable]

[MICROFICHE/COPYRIGHT REFERENCE]

[0004] [Not Applicable]

BACKGROUND OF THE INVENTION

[0005] Embedded systems (i.e., mobile electronic devices having embedded software/firmware), such as mobile cellular phones, personal digital assistants (PDA's),

MP3 players, digital cameras, etc. often contain firmware and/or application software that is either provided by the manufacturers of the electronic devices, telecommunication carriers, or third parties. Mobile electronic devices may be embedded systems because the firmware and application software that they contain may be embedded in silicon memory, such as FLASH. However, firmware and/or application software often contain software bugs. New versions of the firmware and software are periodically released to fix the bugs, introduce new features, or both.

[0006] A fundamental problem in providing access to new releases of firmware and software is that the embedded systems (mobile electronic devices) are often constrained in terms of resources, such as available memory. Attempts to upgrade firmware or software by end-users often results in making the device, or some features of the device inoperable. Additionally, when changes to a firmware component, such as a driver, need to be made, there is no easy way to swap the old one out and put in a new one. Typically, the device has to be reflashed by a new FLASH image that is created by recompiling all or most of the code and/or rebuilding the image file. It can be seen that there is a need for an architecture where such recompilation or rebuilding is either made unnecessary or minimized and simplified.

[0007] When a new version of a driver or a function is to be introduced into embedded systems, all references to the old driver or function that occur in all other software/firmware sections have to be changed. Similarly, when a data item is shifted around in the memory of an embedded device, all references to that particular data item need to be updated. These are at least two issues that must be faced by embedded system designers.

[0008] In current embedded system technology, for example in mobile electronic devices, firmware is one big mass making it very hard to manipulate or update individual functions or sections independently. It can be seen that there is a need for a next generation architecture that enables updates of independent components within mobile electronic devices having embedded systems.

[0009] Further limitations and disadvantages of conventional and traditional approaches will become apparent to one of skill in the art, through comparison of such systems with

some aspects of the present invention as set forth in the remainder of the present application with reference to the drawings appended hereto.

SUMMARY OF THE INVENTION

[0010] Aspects of the present invention may be found in a method and apparatus for designing modular components for software and firmware employed in embedded systems. Aspects of the present invention also provide a method and system for generating, distributing and applying update packages for modular, componentized firmware/software upgrades in embedded systems.

[0011] A method according to an embodiment of the present invention comprises a method of compiling code executable in an electronic device having a component architecture platform (CAP) framework. The method comprises compiling code targeted to the CAP framework, creating a reference lookup table, and saving compiled code and the reference lookup table.

[0012] In yet another embodiment of the present invention compiling code comprises generating program instructions that use symbolic references for at least one of program variables, functions, methods, and operands, and extracting reference information and populating the reference lookup table with extracted reference information.

[0013] In another embodiment of the present invention creating a reference lookup table comprises associating at least one symbolic reference with a physical address, and maintaining the reference information in entries. The entries map symbolic references encountered in the code during compilation to memory addresses.

[0014] In another embodiment of the present invention the entries comprise address values assigned to references corresponding to one of a compiler phase and a link/load phase of the compilation.

[0015] In another embodiment of the present invention the method further comprises generating one of binary and hex output for compiled code that create entries corresponding to at least one entry in the reference lookup table.

[0016] In another embodiment of the present invention the compiled code is adapted for executing individual program instructions and the individual program instructions are pre-fetched into a pipeline of instructions in preparation for execution.

[0017] In another embodiment of the present invention processing time is reduced by resolving references via a reference lookup table management unit, and the unit facilitates resolution of the references for pre-fetching instructions into the pipeline.

[0018] Another method according to an embodiment of the present invention comprises a method of generating update packages for updating software in a mobile electronic device capable of employing the component architecture platform (CAP) framework. The method comprises retrieving an existing and an updated version of code, determining which program components to modify, generating an update package having modules corresponding to those program components to be modified and generating an associated reference lookup table having entries corresponding to those program components to be modified.

[0019] In another embodiment of the present invention determining which program components to modify comprises determining program components to be one of left unchanged, deleted, added, and modified. Program components left unchanged are not included in the update package.

[0020] In another embodiment of the present invention the update package and the associated reference lookup table modifications are adapted for transfer to an embedded system in the electronic device as one of a single program unit and two different related program units transferred when the electronic device is updated.

[0021] In another embodiment of the present invention the update package is adapted to facilitate update by an update agent in the electronic device.

[0022] In another embodiment of the present invention the update package comprises information for adding new modules. The update package replacing existing modules with new modules and updating the associated reference lookup table.

[0023] A method in accordance with an embodiment of the present invention comprises a method of operating a processor supporting a component architecture platform (CAP) framework. The method comprises determining an initial processor execution mode, setting the initial processor execution mode, executing program instructions, and communicating with a reference lookup table management unit.

[0024] In another embodiment of the present invention the method further comprises looking up additional references, and populating a pipeline with at least one of program instructions, references, and addresses.

[0025] In another embodiment of the present invention the method further comprises initializing a reference lookup table with symbolic reference values to be resolved, and retrieving addresses for information corresponding to the symbolic reference values.

[0026] A device in accordance with an embodiment of the present invention comprises a mobile electronic device having an embedded system capable of employing a component architecture platform (CAP) framework. The device comprises a processor, a reference lookup table management unit, primary memory and secondary memory, and at least one reference lookup table.

[0027] In another embodiment of the present invention the device further comprises non-volatile memory, volatile memory, and software resident in at least a portion of primary and secondary memory.

[0028] In another embodiment of the present invention the processor executes program instructions retrieved from at least one of primary and secondary memory.

[0029] In another embodiment of the present invention executing a program instruction comprises performing at least one reference lookup, the at least one reference lookup using symbolic names for one of operands, modules, method names, functions, and components, and the at least one reference lookup may be resolvable into an address at program runtime.

[0030] In another embodiment of the present invention at least one of primary and secondary memory comprises FLASH memory.

[0031] In another embodiment of the present invention the at least one reference lookup table comprises a local reference lookup table mapping at least one of local variables, functions, and methods.

[0032] In another embodiment of the present invention modifications to program instructions are reflected and recorded in the local reference lookup table.

[0033] In another embodiment of the present invention modifications reflected in the local reference lookup table are also reflected in an non-local reference lookup table.

[0034] Another aspect of the present invention comprises a computing environment with a processor capable of executing program instructions from a pipeline. The environment comprises a reference lookup table that is capable of being populated with symbolic references and corresponding location values and a reference lookup table management unit that is capable of managing the reference lookup table and resolving symbolic references in the program instructions into location values before population into the pipeline. The processor is capable of retrieving the program instructions and resolved symbolic references and executing the program instructions.

[0035] In another embodiment of the present invention location values in the reference lookup table are one of memory addresses and offsets.

[0036] These and various other advantages and features of novelty which characterize the invention are pointed out with particularity in the claims annexed hereto and that form a part hereof. However, for a better understanding of the invention, its advantages, and the objects obtained by its use, reference should be made to the drawings which form a further part hereof, and to accompanying descriptive matter, in which there are illustrated and described specific examples of an apparatus in accordance with the invention.

BRIEF DESCRIPTION OF THE DIAGRAMS

[0037] Fig. 1 illustrates block diagram of an embedded system employing component architecture platform (CAP) framework that facilitates easier updates of embedded system software, such as firmware or applications in an electronic device (cellular phones, PDA's, etc.) according to an embodiment of the present invention;

[0038] Fig. 2A illustrates a flow chart of compiler activities for compiling code that may be executed in an embedded system with CAP architecture according to an embodiment of the present invention;

[0039] Fig. 2B illustrates a flow chart of a process for generating update packages for updating firmware/software in an embedded system capable of employing CAP architecture from one version to another version according to an embodiment of the present invention; and

[0040] Fig. 3 illustrates an execution model for a processor in an embedded system capable of executing plural modes and resolving symbolic references in instructions at runtime into addresses of modules according to an embodiment of the present invention.

DETAILED DESCRIPTION OF THE INVENTION

[0041] In the following description of the present invention, reference is made to the accompanying drawings which form a part hereof, and in which is shown by way of illustration an embodiment in which the invention may be practiced. It is to be understood that other embodiments may be utilized as structural changes may be made without departing from the scope of the invention.

[0042] Aspects of the present invention provide a method and apparatus for designing modular components for software and firmware employed in embedded systems. Aspects of the present invention also provide a method and system for generating, distributing, and applying update packages for modular, componentized firmware/software upgrades in embedded systems.

[0043] Fig. 1 is a block diagram of an embedded system 105, employing, for example, a component architecture platform (CAP) framework, that comprises a processor 107, reference lookup table management unit 109, primary memory 113, reference lookup table 111, and secondary memory storage 115, in accordance with the present invention. In Fig. 1, embedded system 105 facilitates easier updates of embedded system software, such as firmware or applications in a mobile electronic device.

[0044] In one embodiment, the embedded system 105 employing a component architecture platform (CAP) framework comprises at least one reference lookup table, non-volatile memory (such as FLASH) in a primary memory 113, volatile memory (such as RAM), software resident in memory, and secondary memory storage 115. The processor 107 executes program instructions that may be retrieved from primary memory 113 or a secondary memory 115, and the program instructions may require reference lookups. A component architecture platform supports replacement, addition, or deletion of software function, by using symbolic rather than physical memory addresses in program instructions. The symbolic addresses are resolved at runtime to physical memory addresses using a reference lookup table.

[0045] The resolution of symbolic addresses to physical memory addresses may be performed in a reference lookup table management unit. The reference lookup table

management unit may provide to the associated processor the physical memory address corresponding to a symbolic address contained in a program instruction. The reference lookup table management unit may be implemented within a processor (hardware) or in code (software).

[0046] In general, references are symbolic names for operands, modules, method names, functions, components, etc. that need a runtime lookup to be resolved into addresses. Addresses for references may be absolute or relative. The ability to look up addresses at runtime provides flexibility to relocate associated modules/functions/components, etc. In general, it may be assumed that invocation of branch-link instructions, jump commands, subroutine calls, and etc. will be preceded by look up of an address from a reference lookup table.

[0047] In another embodiment, a process of loading a new module/function/component into an image of an embedded system is accompanied by activities which may update a reference lookup table, such as reference lookup table 111, with appropriate values for addresses of new or modified functions/components/modules, etc.

[0048] In another embodiment, every module (component/function/method, etc.) may maintain a local reference lookup table, such as reference lookup table 111, that provides mapping of local (internal) variables, functions, and methods. When a new module is inserted into an embedded system, a local reference lookup table may also be inserted. Subsequent modifications to a module may be accompanied by changes to the local reference lookup table, in addition to updates to an overall reference lookup table that maintains non-local reference lookup information.

[0049] In another embodiment of the present invention, an embedded system 105 may employ chipsets that facilitate maintenance and management of reference lookup tables. The chipsets, such as reference lookup table management unit 109, may employ micro-coding techniques to implement reference table lookups, reference table management, and access to references during execution of code at runtime. Software development tools such as compilers that compile applications, may generate binary or hex output for compiled code that create entries to the reference lookup tables, and may also manipulate tables, such as reference lookup table 111.

[0050] Fig. 2A illustrates a flow chart of compiler activities for compiling code that may be executed in an embedded system having the CAP architecture. At block 207, a compiler compiles code targeted to the CAP architecture. This may involve extracting reference information and maintaining the reference information. Later, at block 209, a reference lookup table may be created and populated with entries of extracted reference information. The entries may be a map of symbolic names of references encountered in the code during compilation, and address values (relative or absolute) that may be assigned to entries in a compiler phase or in a subsequent link/load phase (when the references may be loaded into memory for execution). At block 211, the compiled code may be saved along with an associated reference lookup table.

[0051] Actual compiling activity may be executed by the embedded system or by another system that targets the embedded system for execution. The compiled code may be executed in an embedded system. The processor may execute the code by retrieving and executing individual program instructions, often in a specific order. The program instructions may be retrieved ahead of an execution order and a pipeline of instructions may be maintained, with references for the pipeline of instructions resolved into addresses or other values in preparation for execution of the instructions.

[0052] By pre-fetching instructions into a pipeline of instructions and by resolving references before it is time to execute the instructions, the processor may avoid spending additional time in resolving addresses or values associated with the references. The reference lookup table management unit 109 may facilitate resolution of the references for program instructions in the pipeline.

[0053] Fig. 2B illustrates an exemplary process of generating update packages for updating firmware/software in an embedded system from one version to another version and a process of retrieving and updating the embedded system, in accordance with an embodiment of the present invention. The embedded system may employ a CAP architecture. At block 221, new and old versions of code, such as a collection of code loaded into an embedded system, may be retrieved by a generator environment, for example. The generator environment may determine what modules/component/functions may be added, deleted, or modified between the old and the new versions. Only modules

that are added, deleted, or modified may be considered as candidates for inclusion in an update package.

[0054] At block 223, an update package may be generated and reference lookup table modifications may be determined. Modifications to the reference lookup table may be associated with modules/components/functions that are added, deleted, or modified between the old and new versions.

[0055] At block 225, an update package may be transferred to the embedded system along with associated reference lookup table modifications. In one embodiment, the update package may comprise the reference lookup table modifications. In another related embodiment, reference lookup table modifications and the update package may be treated as two different but related units of data to be transferred to the embedded system.

[0056] At block 227, the update agent in the embedded system processes the retrieved update package and associated reference lookup table modifications. The embedded system loads new modules (methods, functions, subroutines, classes, etc.) if any, for the new version of the embedded system being in the update package. The new modules replace existing modules in the embedded system. The update agent in the embedded system also applies modifications to those modules that are to be modified but not replaced. These activities may be only part of an update process facilitated by an update agent in an embedded system.

[0057] References to newly added modules, modified modules, or deleted modules may be updated in a reference lookup table at block 229. Processing stops at block 231 when the update process is completed.

[0058] Fig. 3 illustrates an execution model for an exemplary processor, such as processor 107 of embedded system 105, in accordance with an embodiment of the present invention. The processor may execute in one of at least two modes: a reference lookup table (RLT) mode, where the processor may resolve at runtime symbolic references in instructions into addresses of modules or operands, etc.; and a regular mode, where the processor may execute program instructions in which references have already been resolved, and for which a reference lookup table is not employed.

[0059] At block 321, the processor may determine whether to execute program instructions in RLT mode and may set the execution mode to RLT, if not yet set . At block 323, the processor may send a message to a reference lookup table management unit, such as reference lookup management unit 109 disclosed in Fig. 1, to indicate whether to lookup addresses/values for symbolic references in program instructions. The reference lookup table management unit may initialize the reference lookup table, if necessary, with name/value pairs and other related information, wherein each name may be a symbolic reference to be resolved in a current program context.

[0060] At block 327, the processor may initiate a pipeline of program instructions to be executed and reference lookup table management unit may be employed to retrieve addresses for all references and populate the references into the pipeline to prepare the pipelined instructions for execution by the processor.

[0061] At block 329, the processor may look up any additional references that may not yet have been resolved and may retrieve addresses or values for the additional references. At the conclusion of execution of program instructions within the current program context, the processor may switch modes to regular execution mode from RLT mode.

[0062] Although a system and method according to the present invention has been described in connection with a preferred embodiment, it is not intended to be limited to the specific form set forth herein, but on the contrary, it is intended to cover such alternatives, modifications, and equivalents, as can be reasonably included within the spirit and scope of the invention as defined by this disclosure and the appended diagrams.